# A quadratic Welch-Berlekamp algorithm to decode generalized Gabidulin codes, and some variants

Gwezheneg Robert

# A quadratic Welch-Berlekamp algorithm to decode generalized Gabidulin codes, and some variants

Gwezheneg ROBERT
IRMAR, Université de Rennes 1
Rennes, France
Email: gwezheneg.robert@univ-rennes1.fr

***Abstract*—Gabidulin codes are Maximum Rank Distance (MRD) codes. They have been recently generalized to cyclic Galois extension fields. The unique decoding problem is equivalent to the linear reconstruction problem.**

**The aim of this article is the study of an algorithm to solve this reconstruction problem. We prove that the output of our algorithm is a solution of the reconstruction problem. Then we give some variants.**

**We also establish that (one of the variant of) the algorithm is quadratic.**

***Index Terms*—Gabidulin codes, rank metric, skew polynomials, decoding algorithm.**

## I. Motivation

Gabidulin codes have been discovered in 1978 [4], in 1985 [5] then in 1991 [12]. In Gabidulin's article, the codes are defined similarly to Reed-Solomon codes. The difference between these two kinds of codes is the metric and the evaluated polynomials. Thus, they have some common properties, like their optimality concerning the Singleton bound. This optimality explains the various uses of Gabidulin codes.

For example, in network coding, interesting codes are designed from Gabidulin codes with the lifting construction. Such codes are also used for distributed storage [10].

Binary Gabidulin codes have been used to design space-time code [3] [7]. The main difficulty was to get codewords in the complex field. For this purpose, Gabidulin codes have been generalized to cyclic Galois extensions of the rational field [1] [11].

Nevertheless, a large error-correcting capability is useless without an efficient decoding algorithm. Several algorithms have been proposed, most of them coming from Reed-Solomon decoders. Among all these algorithms, we choose the Welch-Berlemkamp-like one [6] for several reasons. First of all, the algorithm has a quadratic complexity. Then, it is easy to implement and requires few memory. Last reasons come from the study of the algorithm and confirm this choice. A division-free variant enables the use of this algorithm for codes over integer rings [11]. Then, there is no conditions on the support of the Gabidulin code, which enables the adaptation of the algorithm for decoding erasures [11].

## II. Contribution

The algorithm we study is inspired by the Welch-Berlekamp algorithm [2], adapted to Gabidulin codes [6]. We focus on the reconstruction problem, which is the main part of the decoder. There are some cases for which the algorithm of [6] does not work. Namely, the algorithm fails if a discrepancy ($u_0$ or $u_1$) is zero. In this article, we fill the remaining gaps and prove that the output is a solution of the reconstruction problem. Moreover, we give three variants. We also establish the complexity of the algorithm.

Section III gives definition and properties of generalized Gabidulin codes. Section IV introduces a decoding method, based on the reconstruction problem. Proofs are omitted due to brevity (see [1], [11] and [6]). Contribution concerns the reconstruction algorithm (Algorithm 2). In Section V, we prove that the output of the algorithm is a solution of the reconstruction problem. We also give the algebraic complexity of this algorithm. Section VI is dedicated to the three variants of the algorithm.

## III. Generalised Gabidulin codes

In this section, we generalise the definition of Gabidulin codes to cyclic Galois extensions. We first introduce $\theta$-polynomials and rank metric.

In the whole article, we consider a cyclic Galois extension $K \hookrightarrow L$ of degree $m$, and a generator $\theta$ of the Galois group of the extension. (We directly choose convenient extensions, which fill the conditions required for Theorems 1 and 4.) Let $\mathcal{B} = (b_1, \dots, b_m)$ be a $K$-basis of $L$ (as a $K$-vector space). The following mapping enables to turn an element of $L$ into a vector of $K^m$.

$$\begin{array}{rccc} \mathrm{ext}_{\mathcal{B}} : & L & \longrightarrow & K^m \\ & x = x_1 b_1 + \cdots x_m b_m & \longmapsto & (x_1, \dots, x_m) \end{array}$$

### A. $\theta$-polynomials

$\theta$-polynomials are a particular case of skew polynomials [9]. They are the generalisation of $q$-polynomials [8] used for original Gabidulin codes.

**Definition 1** ($\theta$-polynomials)**.**

- *A $\theta$-polynomial with coefficients in $L$ is an element of the form*

$$\sum_{i \geqslant 0} a_i X^i : \ a_i \in L,$$

  *with a finite number of non-zero $a_i$.*
- *The degree of a non-zero $\theta$-polynomial $\sum_{i \geqslant 0} a_i X^i$ is* $\deg(A) = \max\{i : a_i \neq 0\}$. *(By convention, the degree of the zero $\theta$-polynomial is $-\infty$.)*

- *The set of $\theta$-polynomials with coefficients in $L$ is denoted by $L[X;\theta]$.*

We can define some operations over $\theta$-polynomials.

**Definition 2** (Operations over $\theta$-polynomials)**.** *Let $A = \sum_{i=0}^{n} a_i X^i \in L[X;\theta]$, $B = \sum_{i=0}^{m} b_i X^i \in L[X;\theta]$, and $c \in L$.*

- *The addition is: $A + B = \sum_i (a_i + b_i) X^i$.*
- *The (symbolic) product is defined by: $X \cdot c = \theta(c) \cdot X$. Thus the product is well-defined: $A \cdot B = \sum_{i,j} a_i \theta^i(b_j) X^{i+j}$.*
- *The evaluation is: $A(c) = \sum_i a_i \theta^i(c)$.*

The ring of $\theta$-polynomials is a non-commutative ring. There is a left and a right Euclidean divisions. Notice that in this ring, there are several ways to define an evaluation. We choose the operator evaluation since it generalizes the evaluation used in Gabidulin codes over finite fields.

**Remark 1.** *When we consider finite fields $\mathbb{F}_q \hookrightarrow \mathbb{F}_{q^m}$, provided with the Frobenius automorphism $x \mapsto x^q$, we get the $q$-polynomials used in original Gabidulin codes.*

The algorithm relies on the three following properties.

**Theorem 1.**
- *Let $A$ be a $\theta$-polynomial of degree $d$. Then the set of roots of $A$ (i.e. elements $x$ such that $A(x) = 0$) is a $K$-vector space of dimension at most $d$.*
- *Let $A$ be a $\theta$-polynomial such that $A(v_i) = 0 : 1 \leqslant i \leqslant s$, where $v_1, \ldots, v_s$ are $K$-linearly independent values of $L$. Then either $\deg(A) \geqslant s$ or $A = 0$.*

**Definition 3** (Annihilator $\theta$-polynomial)**.**
- *Let $V$ be a $K$-subspace of $L$ (as a vector space). An annihilator $\theta$-polynomial of $V$ is a monic non-zero $\theta$-polynomial $\mathcal{A}$ of minimal degree such that :*

$$\forall v \in V,\ \mathcal{A}(v) = 0.$$

- *Let $(v_1, \ldots, v_s) \in L^s$. An annihilator $\theta$-polynomial of $(v_1, \ldots, v_s)$ is an annihilator $\theta$-polynomial of $\mathrm{Vect}(v_1, \ldots, v_s)$.*

**Proposition 2.** *Given a $K$-subspace $V$ of $L$, there is a unique annihilator $\theta$-polynomial of $V$. Moreover, its degree is exactly $\dim(V)$.*

**Definition 4** (Interpolating $\theta$-polynomial)**.** *Let $(x_1, \ldots, x_s) \in L^s$ and $(y_1, \ldots, y_s) \in L^s$. An interpolating $\theta$-polynomial $\mathcal{I}$ is a $\theta$-polynomial of minimal degree such that :*

$$\forall 1 \leqslant i \leqslant s,\ \mathcal{I}(x_i) = y_i.$$

**Proposition 3.** *If $x_1, \ldots, x_s$ are $K$-linearly independent, then there exists a unique interpolating $\theta$-polynomial of degree at most $s - 1$.*

### *B. Rank metric*

**Definition 5** (Rank metric)**.** *Let $x = (x_1, \ldots, x_s) \in L^s$. The rank weight of $x$ is the rank of the matrix whose columns are the column vectors $\mathrm{ext}_{\mathcal{B}}(x_i)$ :*

$$w_r(x) = \mathrm{rank}\left(\mathrm{ext}_{\mathcal{B}}(x_1), \cdots, \mathrm{ext}_{\mathcal{B}}(x_s)\right).$$

*The rank distance between $x$ and $y$ is $d_r(x, y) = w_r(x - y)$.*

There is another way to define this metric.

**Definition 6** (Alternative rank metric)**.** *Let $x = (x_1, \ldots, x_s) \in L^s$. The rank weight $w_{r'}(x)$ of $x$ is the degree of the annihilator $\theta$-polynomial of $\mathrm{Vect}(x_1, \ldots, x_s)$.*
*The rank distance between $x$ and $y$ is $d_{r'}(x, y) = w_{r'}(x - y)$.*

**Theorem 4.**

$$\forall x \in L^s,\ w_r(x) = w_{r'}(x)$$

*Thus, the two metrics are identical and called* the *rank metric.*

**Proposition 5** (Singleton bound)**.** *Let $\mathcal{C}$ be an error correcting code of length $n$, dimension $k$. Then its minimal distance $d$ (in rank metric), satisfies : $d \leqslant n - k + 1$.*

**Definition 7** (MRD codes)**.** *Let $\mathcal{C}$ be an error correcting code of parameters $[n, k, d]$. The code is MRD (Maximal Rank Distance) if $d = n - k + 1$.*

### *C. Gabidulin codes*

**Definition 8** (Generalised Gabidulin code)**.** *Let $k \leqslant n \leqslant m$ be integers and $g = (g_1, \ldots, g_n)$ be $K$-linearly independent elements of $L$. The* generalised Gabidulin code $\mathrm{Gab}_{\theta,k}(g)$ *of length $n$, dimension $k$ and support $g$ is*

$$\mathrm{Gab}_{\theta,k}(g) = \{ (f(g_1), \ldots, f(g_n)) \ : \tag{1}$$
$$f \in L[X;\theta], \deg(f) < k \}. \tag{2}$$

*The codeword obtained with the information word $f$ is denoted by $c(f)$.*

**Proposition 6.** *A generalised Gabidulin code is a linear code, whose generator matrix is*

$$\begin{pmatrix} \theta^0(g_1) & \cdots & \theta^0(g_n) \\ \vdots & \ddots & \vdots \\ \theta^{k-1}(g_1) & \cdots & \theta^{k-1}(g_n) \end{pmatrix}.$$

**Proposition 7.** *A generalised Gabidulin code of length $n$ and dimension $k$ has minimal distance (in rank metric) $d = n - k + 1$. Thus, generalised Gabidulin codes are MRD codes, and we can correct an error of rank $t$ if*

$$t \leqslant \left\lfloor \frac{n-k}{2} \right\rfloor.$$

**Example 1.** *Consider the extension $\mathbb{F}_2 \hookrightarrow \mathbb{F}_{2^4} = \mathbb{F}_2[\alpha] = \mathbb{F}_2[Y]/(Y^4 + Y + 1)$, provided with the Frobenius automorphism $x \mapsto x^2$. Let $\mathcal{G}$ be the Gabidulin code of length $n = 4$, dimension $k = 2$ and support $g = \left(1, \alpha, \alpha^2, \alpha^3\right)$. We consider the information word*

$$f = \alpha^5 + \alpha^7 X.$$

*Then, the corresponding codeword is*

$$c(f) = \left(\alpha^{13}, \alpha^5, \alpha^8, \alpha^3\right) \in \mathbb{F}_{16}^4.$$

## IV. Decoding by Reconstruction

The reconstruction problem and its adaptation to the decoding of Gabidulin codes is described in [6]. The method is also valid for Gabidulin codes over infinite fields..

Let $\mathrm{Gab}_{\theta,k}(g)$ be a Gabidulin code. Let $f$ be an information word, and $c(f)$ the corresponding codeword. During the transmission, an error is added, and the receiver gets

$$y = c(f) + e,$$

where $y_i = f(g_i) + e_i$. The goal of the receiver is to recover $f$, knowing $g$ and $y$.

**Definition 9** (Decoding Problem)**.** *Given a Gabidulin code of length $n$, dimension $k$ and support $g$, and a received word $y$, solving the Decoding Problem* $\mathrm{Dec}(n, k, g, y)$ *consists of finding*

- *a $\theta$-polynomial $f \in L[X;\theta]$ of degree* $\deg f < k$*,*
- *a vector $e \in L^n$ of rank $w(e) \leqslant t = \lfloor \frac{n-k}{2} \rfloor$,*

*such that*

$$y_i = f(g_i) + e_i.$$

We can reformulate this problem with only $\theta$-polynomials. This is the linear reconstruction.

**Definition 10** (Linear Reconstruction Problem)**.** *Given integers $k \leqslant n$ and $t \leqslant n$, a vector $g \in L^n$ of $K$-linearly independent values, and a vector $y \in L^n$, solving the Linear Reconstruction problem* $\mathrm{LR}(n, k, t, g, y)$ *consists of finding*

- *a $\theta$-polynomial $N \in L[X;\theta]$ of degree* $\deg N < k + t$*,*
- *a non-zero $\theta$-polynomial $W \in L[X;\theta]$ of degree* $\deg W \leqslant t$*,*

*such that*

$$W(y_i) = N(g_i).$$

Any solution $(f, e)$ of $\mathrm{Dec}(n, k, g, y)$ gives a solution $(N, W)$ of $\mathrm{LR}(n, k, \lfloor \frac{n-k}{2} \rfloor, g, y)$, where $W$ is the annihilator polynomial of $e$, and $N = W \cdot f$. The interesting fact is the reverse property.

**Theorem 8.** *If $t \leqslant \lfloor \frac{n-k}{2} \rfloor$ and if there exists a solution of* $\mathrm{Dec}(n, k, g, y)$*, then any solution of* $\mathrm{LR}(n, k, t, g, y)$ *gives the solution of* $\mathrm{Dec}(n, k, g, y)$*.*

The decoding algorithm is given in Algorithm 1.

Algorithm 1: decoding algorithm

**Input:** dimension $k$ and length $n$ of the code
$(g_1, \ldots, g_n)$, support of the code
$(y_1, \ldots, y_n)$, received word
**Output:** $f$, the information word.

Compute $(N, W)$, solution of $\mathrm{LR}(n, k, t, g, y)$
Compute $Q$ and $R$ such that $N = W \cdot Q + R$ (Euclidean division)
**if** $R = 0$
    **return** $Q$
**else**
    **return** "Failure : too much errors"
**end if**

Algorithm 2: reconstruction algorithm

**Input:** dimension $k$ and length $n$ of the code
error-correcting capability $t$ of the code
$(g_1, \ldots, g_n)$, support of the code
$(y_1, \ldots, y_n)$, received word
**Output:** $N$ and $W$ solutions of the $\mathrm{LR}(n, k, t, g, y)$

# Initialisation
$N_0 \longleftarrow 1$
$N_1 \longleftarrow 0$
$W_0(X) \longleftarrow 0$
$W_1(X) \longleftarrow 1$
**for** $i$ **from** $1$ **to** $k$
    $N_1 \longleftarrow N_1 + \frac{y_i - N_1(g_i)}{N_0(g_i)} \cdot N_0$
    $N_0 \longleftarrow (X - \frac{\theta(N_0(x_i))}{N_0(g_i)}) \cdot N_0$
**end for**
# Main Loop
**for** $i$ **from** $k+1$ **to** $n$
    # Computation of discrepancies
    $u_0 \longleftarrow N_0(g_i) - W_0(y_i)$
    $u_1 \longleftarrow N_1(g_i) - W_1(y_i)$
    # Secondary Loop
    **if** $u_0 \neq 0$ and $u_1 = 0$
        $j \longleftarrow i + 1$
        **while** $u_0 \neq 0$ and $u_1 = 0$ and $j \leqslant n$
            $u_0 \longleftarrow N_0(g_j) - W_0(y_j)$
            $u_1 \longleftarrow N_1(g_j) - W_1(y_j)$
            $j \longleftarrow j + 1$
        **end while**
        **if** $j = n + 1$
            **return** $(N_1, W_1)$
        **else**
            $g_i \longleftrightarrow g_j$
            $y_i \longleftrightarrow y_j$
        **end if**
    **end if**
    # update of $\theta$-polynomials, according to discrepancies
    **if** $u_0 \neq 0$ and $u_1 \neq 0$
        $N_0' \longleftarrow (X - \frac{\theta(u_1)}{u_1}) \cdot N_1$
        $W_0' \longleftarrow (X - \frac{\theta(u_1)}{u_1}) \cdot W_1$
        $N_1' \longleftarrow N_0 - \frac{u_0}{u_1} N_1$
        $W_1' \longleftarrow W_0 - \frac{u_0}{u_1} W_1$
    **end if**
    **if** $u_0 = 0$ and $u_1 \neq 0$
        $N_0' \longleftarrow (X - \frac{\theta(u_1)}{u_1}) \cdot N_1$
        $W_0' \longleftarrow (X - \frac{\theta(u_1)}{u_1}) \cdot W_1$
        $N_1' \longleftarrow N_0$
        $W_1' \longleftarrow W_0$
    **end if**
    **if** $u_0 = 0$ and $u_1 = 0$
        $N_0' \longleftarrow X \cdot N_1$
        $W_0' \longleftarrow X \cdot W_1$
        $N_1' \longleftarrow N_0$
        $W_1' \longleftarrow W_0$
    **end if**
    $N_0 \longleftarrow N_0'$
    $W_0 \longleftarrow W_0'$
    $N_1 \longleftarrow N_1'$
    $W_1 \longleftarrow W_1'$
**end for**
**if** $n - k$ is even
    **return** $N_1, W_1$
**else**
    **return** $N_0, W_0$
**end if**

## V. Proof of correctness

We now focus on the Reconstruction algorithm, given above (Algorithm 2).

The goal of this section is to establish that the output of the algorithm is a solution of the Reconstruction Problem.

**Theorem 9.** *Let $k \leqslant n$ and $t = \lfloor \frac{n-k}{2} \rfloor$ be integer parameters, $g \in L^n$ be $n$ $K$-linearly independent values of $L$ and $y \in L^n$ be a received word. Let $(N, W)$ denote the output of the algorithm with these inputs. Then $(N, W)$ is a solution of the problem $\mathrm{LR}(n, k, t, g, y)$. Namely, we have :*

1) $N(g_i) = W(y_i), 1 \leqslant i \leqslant n$
2) $\deg(N) < k + t$
3) $\deg(W) \leqslant t$
4) $W \neq 0$

The proof is cut in three parts.

**Proposition 10.** *With the notations of Theorem 9, we have $N(g_i) = W(y_i), 1 \leqslant i \leqslant n$.*

*Proof:* We just have to check, by induction, that after step $i$, (during initialisation loop or main loop) $N$ and $W$ verify the induction hypothesis :

$$\mathcal{H}(i) \ : \ \begin{cases} N_0(g_j) = W_0(y_j), 1 \leqslant j \leqslant i \\ N_1(g_j) = W_1(y_j), 1 \leqslant j \leqslant i \end{cases} .$$

At the end of the algorithm (normal output), we have $\mathcal{H}(n)$. In case of premature input (line 39), we have $N_0(g_j) = W_0(y_j), 1 \leqslant j \leqslant i$ by induction hypothesis, and we have $N_0(g_j) = W_0(y_j), i \leqslant j$, which is part of the conditions of premature input. ∎

**Proposition 11.** *With the notations of Theorem 9, we have $\deg(N) < k + t$ and $\deg(W) \leqslant t$.*

*Proof:* During the initialisation loop, degrees of $N_1$ and $N_0$ increase by 1 during each step. During the main loop, they increase as follows:

$$\begin{array}{rcl} \deg(N_0') & = & \deg(N_1), \\ \deg(W_0') & = & \deg(W_1), \\ \deg(N_1') & \leqslant & \max(\deg(N_0), \deg(N_1)), \\ \deg(W_1') & \leqslant & \max(\deg(W_0), \deg(W_1)). \end{array}$$

Thus, we have the following upper bounds :

$$\begin{array}{rcl} \deg(N_0) & \leqslant & k + \lfloor \frac{i-k}{2} \rfloor, \\ \deg(W_0) & \leqslant & \lfloor \frac{i-k+1}{2} \rfloor, \\ \deg(N_1) & \leqslant & k - 1 + \lfloor \frac{i-k+1}{2} \rfloor, \\ \deg(W_1) & \leqslant & \lfloor \frac{i-k}{2} \rfloor. \end{array}$$

At the end of the algorithm (normal output), one of the pairs $(N_0, W_0)$ or $(N_1, W_1)$ verifies the degree conditions. Since this upper bound is increasing along the steps, in case of premature output, we get a pair which verifies degree conditions. ∎

**Proposition 12.** *With the notations of Theorem 9, we have*

$$W \neq 0.$$

*Proof:* First, remark that $W = 0$ implies $N = 0$. Indeed, when $W = 0$, the interpolation conditions become $N(g_i) = 0$, thus $N$ has $n$ roots linearly independent, so $N = 0$. Then, we can list the degree possibilities according to the parity of the step and according to which polynomial reaches the upper bounds. The 8 cases are the following.

- For even steps :
  - all polynomials reach the upper bound
  - only $N_1$ has degree lower than the upper bound
  - only $W_0$ has degree lower than the upper bound
  - both $N_1$ and $W_0$ have degree lower than the upper bound
- For odd steps :
  - all polynomials reach the upper bound
  - only $N_0$ has degree lower than the upper bound
  - only $W_1$ has degree lower than the upper bound
  - both $N_0$ and $W_1$ have degree lower than the upper bound

Then, we never have the pair $(0, 0)$ since two polynomials of the same pair never have lower degree than the upper bound simultaneously. ∎

**Proposition 13.** *The main part of the algorithm (meaning all except the secondary loop) requires $2n^2 + O(n)$ multiplications and $2n$ divisions.*

During the secondary loop, some operations are added, due to computation of discrepancies (lines 34 and 35). In the worst case (meaning that we compute *all* possible discrepancies at *each* step), we add $\frac{1}{3}(n-k)(n^2 + 5nk + 2k^2) + O(n^2)$ multiplications. The third variant of section VI decreases this complexity to 0, making the algorithm quadratic.

**Remark 2.** *The secondary loop enables to stop the algorithm as soon as we have computed a solution. (When the weight of the error is $\tau$, the algorithm stops after $2\tau$ steps.)*

## VI. Variants

The goal of this section is to describe some variants of the algorithm. For each of them, we see the interest and the effect on the complexity.

### A. A division-free algorithm

This variant has interest for codes in number fields. It enables, when the support of the code, the information word and the error have coefficients in the integer ring, to get polynomials whose coefficients are also in the integer ring. We only have to modify updates, which become : (lines that do NOT appear in variants haven't changed)

| Modification for 'division-free' variant | |
|---|---|
| 5 : | $\lambda \leftarrow 1$ |
| 6 : | **for** $i$ **from** 1 **to** $k$ |
| 7 : | $N_1 \longleftarrow N_0(x_i)N_1 + (\lambda y_i - N_1(x_i)) \cdot N_0$ |
| 8 : | $\lambda \longleftarrow N_0(g_i)\lambda$ |
| 9 : | $N_0 \longleftarrow N_0(x_i)(X - \theta(N_0(x_i))) \cdot N_0$ |
| 10 : | **end for** |

| | |
|---|---|
| 52,62 : | $N'_0 \longleftarrow (u_1X - \theta(u_1)) \cdot N_1$ |
| 53,63 : | $W'_0 \longleftarrow (u_1X - \theta(u_1)) \cdot W_1$ |
| 55 : | $N'_1 \longleftarrow u_1N_0 - u_0N_1$ |
| 56 : | $W'_1 \longleftarrow u_1W_0 - u_0W_1$ |

**Proposition 14.** *The algorithm requires $3n^2 + O(n)$ multiplications and $0$ divisions.*

### *B. Lower degree polynomials*

We can remark that $N_0$ and $N_1$ are updated by additions and left multiplications. Thus, their value along the main loop is a polynomial combination of their value after the initialisation, namely the annihilator $\mathcal{A}$ of $(g_1, \ldots, g_k)$ and the interpolating polynomial $\mathcal{I}$ of $(g_1, \ldots, g_k)$ and $(y_1, \ldots, y_k)$. Thus, we have

$$N_\ell = P_\ell \, \mathcal{A} + W_\ell \, \mathcal{I} \, .$$

$P_\ell$ is updated by the same formulas as $N_\ell$ and $W_\ell$.

We have to modify the computation of the discrepancies.

| Modification for 'lower-degree' variant | |
|---|---|
| 23,34 : | $u_0 = P_0(\mathcal{A}(g_i)) + W_0(\mathcal{I}(g_i) - y_i)$ |
| 24,35 : | $u_1 = P_1(\mathcal{A}(g_i)) + W_1(\mathcal{I}(g_i) - y_i)$ |

The last modification is part of the decoding algorithm but not of the reconstruction algorithm, since it concerns the final division $W \backslash N$ of $N$ by $W$. This division becomes $W \backslash (P \, \mathcal{A}) + \mathcal{I}$.

**Proposition 15.** *The number of multiplication is decreased by $1.5k(n-k)$ for the basic algorithm and by $3.5k(n-k)$ for the division-free variant.*

### *C. Updating discrepancies*

Instead of computing two discrepancies at each step, we can initialise them after the initialisation of polynomials, and update them at each step. The discrepancies $u_0$ and $u_1$, now denoted $u_{0,j}$ and $u_{1,j}$, since they depend on the pair $(g_j, y_j)$, are updated by the same formulas than $N_0$, $W_0$, $N_1$ and $W_1$. Thus, the following lines are added at the initialisation

| Modification for 'updating discrepancies' variant | |
|---|---|
| 11 : | **for** $j$ **from** $k+1$ **to** $n$ |
| 12 : | $u_{0,j} \longleftarrow N_0(g_j) - W_0(y_j)$ |
| 13 : | $u_{1,j} \longleftarrow N_1(g_j) - W_1(y_j)$ |
| 14 : | **end for** |

Computations of discrepancies (lines 23, 24, 34 and 35) are removed.

Then, updates become

| Modification for 'updating discr.' variant | |
|---|---|
| 54, 64 : | $u_{0,j} \longleftarrow (X - \frac{\theta(u_1)}{u_1}) \cdot u_{1,j}$ |
| 57 : | $u_{1,j} \longleftarrow u_{0,j} - \frac{u_0}{u_1} u_{1,j}$ |
| 74 : | $u_{0,j} \longleftarrow X \cdot u_{1,j}$ |
| 67, 77 : | $u_{1,j} \longleftarrow u_{0,j}$ |

The advantage of this variant is that the secondary loop doesn't require computation. Thus, the algorithm is always quadratic.

**Proposition 16.** *The main part of the algorithm (meaning all except additional computations from the secondary loop) has the same complexity for the basic algorithm. In the division-free variant, there are $O(n^2 - k^2)$ additional multiplications. In both cases, the number of computation in the secondary loop is decreased to $0$, thus the algorithm is always quadratic.*

## VII. Conclusion

In this paper, we have described an algorithm solving the reconstruction problem, which is the main step of the decoding of generalised Gabidulin codes. We have proved that the output of the algorithm is a solution of the reconstruction problem. We have adapted the algorithm to integer rings with the 'division-free' variant.

We have also established that the algorithm is quadratic. Indeed, the main part of the algorithm (lines 1 to 29 and 50 to 95) requires a quadratic number of operations, in all variants. In the basic algorithm (meaning Algorithm 2 without variant), the secondary loop (lines 30 to 49) is cubic in the worst case. The 'updating discrepancies' variant enables to decrease the number of these additional computations to $0$, making the whole algorithm always quadratic.

Another improvement concerning the complexity is given. The 'lower degree' variant enables to decrease the constants hidden in the $O$-notation.

## References


[1] Daniel Augot, Pierre Loidreau, and Gwezheneg Robert. Rank metric and Gabidulin codes in characteristic zero. In *ISIT 2013 IEEE International Symposium on Information Theory*, 2013.

[2] Elwyn R Berlekamp and Lloyd R Welch. Error correction for algebraic block codes, December 30 1986. US Patent 4,633,470.

[3] M Bossert, EM Gabidulin, and P Lusina. Space-time codes based on Gaussian integers. In *Information Theory, 2002. Proceedings. 2002 IEEE International Symposium on*, page 273. IEEE, 2002.

[4] Ph Delsarte. Bilinear forms over a finite field, with applications to coding theory. *Journal of Combinatorial Theory, Series A*, 25(3):226–241, 1978.

[5] Ernest Mukhamedovich Gabidulin. Theory of codes with maximum rank distance. *Problemy Peredachi Informatsii*, 21(1):3–16, 1985.

[6] Pierre Loidreau. A Welch–Berlekamp like algorithm for decoding Gabidulin codes. In *Coding and Cryptography*, pages 36–45. Springer, 2006.

[7] H-F Lu and PY Kumar. A unified construction of space-time codes with optimal rate-diversity tradeoff. *Information Theory, IEEE Transactions on*, 51(5):1709–1730, 2005.

[8] Oystein Ore. On a special class of polynomials. *Transactions of the American Mathematical Society*, 35(3):559–584, 1933.

[9] Oystein Ore. Theory of non-commutative polynomials. *Annals of mathematics*, pages 480–508, 1933.

[10] Ankit Singh Rawat, Onur Ozan Koyluoglu, Natalia Silberstein, and Sriram Vishwanath. Optimal locally repairable and secure codes for distributed storage systems. *Information Theory, IEEE Transactions on*, 60(1):212–236, 2014.

[11] Gwezheneg Robert. *Codes de Gabidulin en caractéristique nulle. Application au codage espace-temps.* PhD thesis, Université de Rennes 1, 2015.

[12] Ron M. Roth. Maximum-rank array codes and their application to crisscross error correction. *Information Theory, IEEE Transactions on*, 37(2):328–336, 1991.